Dynamic Methods for Fragment Assembly in Large-Scale Genome Sequencing Projects

Wendy Istvanick* Amy Kryder† Gary Lewandowski‡ João Meidânis§ Anton Rang
Stacia Wyman¶ Deborah Joseph‖

Computer Sciences Department, University of Wisconsin-Madison, Madison WI 53706, USA

Abstract

Large-scale genome sequencing projects present a unique set of problems not found in smaller sequencing efforts. Key to the success of large sequencing efforts, such as the Human Genome Program, is the mathematical and computational tools for organizing and analyzing large quantities of genetic sequence data. In this paper, we describe a package for handling fragment assembly problems that arise in large-scale sequencing projects employing random (shotgun) strategies. We have developed a package of dynamic data structures and algorithms for assembling and maintaining fragments. The system is unique in that it maintains the layout information for the fragment assembly in a set of dynamic data structures that permit new data to be very quickly added and analysis to be carried out at any point in the assembly process.

The work described arises from co-operative research with the *E. coli* Genome Project at the University of Wisconsin.

1 Introduction

Large-scale genome sequencing projects present a unique set of problems not found in smaller sequencing efforts. Key to their success are mathematical and computational tools for organizing and analyzing large quantities of genetic sequence data. This is particularly true when random techniques, such as shotgun sequencing, are used. The goal of our research is to provide a fragment assembly package geared directly to the problems encountered by large sequencing projects that employ random (shotgun) strategies.

Current technology limits the direct sequencing of DNA to fragments of approximately 500 bases. When sequencing longer strands it is necessary to break the long sequences into many short fragments, sequence these fragments, and from them reconstruct the longer sequence. In large genome projects, this is actually a multi-level process. For instance, in the *E. coli* genome project at the University of Wisconsin, the entire bacterial genome (a single circular chromosone) is cut into several hundred pieces ranging in size from 15 to 20 thousand bases (the *λ-clones*). These cuts are precise and occur at well defined locations, such as restriction enzyme sites. Each λ-clone is broken up into fragments 500 to 1000 bases in length by a random process (sonication or French press). These fragments are short enough to permit direct sequencing. Once sequenced, the fragment data is *assembled* to reconstruct the information for each λ-clone. If the random process of fragmentation leaves gaps in the coverage of a λ-clone, either more random fragments are sequenced, or directed methods (eg. vector flipping or PCR) are used to sequence across the gaps.

The cost of sequencing projects is significantly influenced by the accuracy of fragment assembly programs and the ease with which researchers can interact with the alignment program to add additional data, obtain feedback on the sequencing process, edit the alignment, and analyze the consensus sequence for features of biological significance. This is because much of the post alignment analysis and editing must be done by highly trained researchers. The package that we have developed is geared to large projects where data is pro-

*Supported by the *E. coli* Genome Project, F. Blattner's Laboratory,Univ. of Wisc. – Madison.

†Supported by the National Research Service Award 1T32GM08349 from the National Institute of General Medical Sciences.

‡Supported by the Wisconsin Alumni Research Foundation.

§Computer Science Dept., State University of Campinas, Cx. Postal 6065, 13081 – Campinas – SP, Brazil. Partially supported by FAPESP, Brazil, under grant 87/0197-2.

¶Supported by a Univ. of Wisc. – Madison, Computer Sciences Dept. 1st year graduate student summer fellowship.

‖Supported by NSF PYI grant DCR-8451387. Part of this research was done at the Aspen Center for Physics Workshops on Recognizing Genes and Other Components of Genomic Structure, June 1991 and June 1992.

duced, assembled, and analyzed as an ongoing process.

In the next section (Section 2), we describe the overall structure of our dynamic alignment package. Subsections briefly describe some of the more interesting algorithmic aspects of the package. Section 3 discusses implementation issues and recounts some of our experience parallelizing portions of the package. Ongoing work is discussed in Section 4 and an example of an alignment produced by our package is presented in the appendix.

2 A Dynamic Alignment Package for Fragment Assembly

Our fragment assembly package consists of a collection of modules that can interact as subroutines of a main program or as individual programs that pass information through files. Figure 1 depicts the interaction among the modules.

In overall design our package works much as other fragment assembly packages. Initially, in our prepass phase, fragments are compared pairwise to determine which fragments exhibit similarity. Fragments are then grouped into clusters of seemingly similar fragments. At this point ambiguous fragments may be placed in more than one cluster. A layout data structure is then constructed for each cluster. The layout data structure is the primary data structure of the entire fragment assembly package and the structure on which all subsequent processing occurs. These structures can be merged, consensus sequences can be computed from them, and statistical analysis can be done on the sequences in structures.

Below we give a somewhat more detailed description of each module.

2.1 Preprocessing the Fragments

The first phase of the alignment process, called *prepass*, performs a pairwise comparison for every pair of fragments (and their reverse complements) to compute an estimate of pairwise similarity. This calculation is done by sliding a window of fixed size across each fragment and hashing the words encountered into a binary hash table. No attempt is made to resolve collisions in this hashing process. Thus, for each fragment (and its complement) a bit vector representing the hash table is constructed.

To find the overlap measure for a pair of fragments, the corresponding bit vectors are logically ANDed and the total number of bits set to 1 in the resulting table is computed. A back prediction method, based on the method of moments, is then applied to estimate the number of common windows in the pair of fragments.

Hashing methods have been used in several other contexts for sequence comparison. They were first

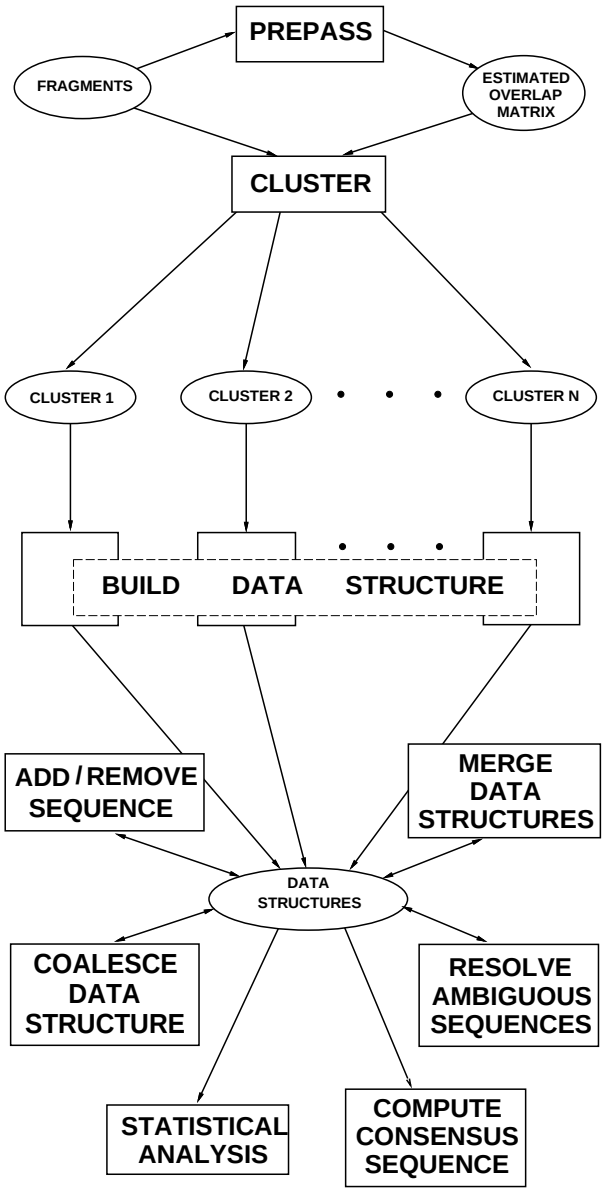


Figure 1: Overview of Dynamic Alignment Package

used by Dumas and Ninio ([5]) and subsequently used by Wilbur and Lipman ([33]). Similar methods, using common subwords are used in FASTA and related programs ([18]). A study of the speed and effectiveness of FASTA-like methods can be found in [19] and [20]. In [10], we present an empirical study, as well as, a theoretical derivation of the back prediction method used in our prepass strategy.

The output from the prepass module is used to isolate fragments with significant overlap and reduce the problem by screening out dissimilar fragments, obviating the need to try and align fragments that are completely unrelated.

2.2 Grouping Fragments into Clusters

The next step in the process is to *cluster* like fragments. This is done by constructing a fragment layout using a range tree type representation of an interval graph. An independent set is calculated for this graph. The elements of the independent set are used to cluster the fragments. A cluster is formed for each element of the independent set and consists of all fragments that show sufficient similarity to the independent set member as computed by the prepass program.

Clustering allows us to gracefully handle the problem of projects that initially assemble into multiple contigs due to lack of coverage and permits us to partition the data both to run in parallel and to run on smaller machines.

The use of interval graphs in computational genetics is hardly new. For the fragment assembly problem, the algorithms of Peltola et al. [21] were perhaps the first to explicitly use the properties of interval graphs. Our representation of the graphs using trees of the form encountered in computational geometry problems is new. The manner in which the trees are constructed provides the algorithm with a very nice feature. As each fragment (or its complement) is inserted in the tree we verify the consistency of the overlap information computed by the prepass process. In the case of repetitive fragments, an inconsistency will appear at this point. Our algorithms allow us to flag such fragments and insert them in more than one cluster. After the layout data structure is constructed and additional alignment information is available, the fragment can be retained in the location where it aligns best and removed from other locations.

2.3 Ordering Fragments within Clusters

The fragments in each cluster must be linearly ordered for construction of the fragment layout data structures. The ordering used is obtained from a maximum spanning tree algorithm applied to each cluster.

2a: Two Aligned Fragments

```
T  C  C  A  A  A  T  C  G  A
T  T  C     A  A  T  G  C  A
```

2b: Layout Data Structure

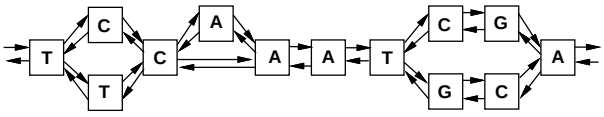


Figure 2: Layout Data Structure for Two Fragments

A point worth noting is that while the maximum spanning tree is unique in a theoretical sense, in a practical sense it is not. The similarity matrix computed by the prepass does not give an exact overlap between fragments. Furthermore, different methods of computing the maximum spanning tree result in different linear orderings of the fragments in a cluster. We have implemented two of the standard methods: building the tree out from a single root and building the tree by constructing small trees that merge. Our initial results show that when coverage is poor the later method results in better layouts.

2.4 Constructing the Layout Data Structures

The layout data structure is the primary data structure in the system. It is illustrated pictorially in Figure 2. This figure shows two fragments which have been aligned and put into a linked list such that common bases share nodes of the structure.

All of the fragments from a cluster are inserted into a single layout data structure using the ordering produced earlier. Each fragment is inserted relative to the fragment it is closest to in the maximum spanning tree. A module similar to FASTA ([18]) produces co-ordinates of exact overlap between the fragments. Subfragments which align exactly share nodes in the data structure.

The nodes of the data structure have a rather complicated structure. They contain information about the fragments and also information that aids in traversing the structure and computing statistical information about the alignment. We can also use information stored in the nodes to distinguish between fragments from the assembly project and fragments that a user may insert for other reasons. For instance, it may be useful to insert fragments obtained from a genome database. These fragments may be useful for

comparison or to pull together contigs with low coverage, but they must be identified so that they do not influence the consensus sequence.

Two important things should be noted about the layout data structures. First, the layout data structure captures many of the overlaps between sequence fragments however many overlaps may go undetected. Second, while the data structure does provide an alignment of the fragments, we do not view the data structure as a full base-by-base alignment of all sequences. Instead, further procedures are used to modify and refine the alignment to produce a consensus sequence. These include the coalescing and merging procedures and a multiway alignment procedure for computing consensus sequences.

The collection of modules described below all use the layout data structure.

2.5 Coalesce and Merge

These procedures allow us to merge two layout data structures that share fragments and to further align regions of the data structures. The problems solved by both of these procedures are in theory NP-complete. Thus, the algorithms that we use are heuristics. The coalescing algorithm combines a topological traversal of the layout data structure with a hashing method similar to the prepass algorithm. The merging algorithm relys on common sequences from two layout data structures to produce a combined structure.

2.6 Producing a Consensus Sequence

A unique feature of our assembly package is that the alignment produced in the layout data structure does not represent a consensus sequence. Instead a further procedure is used to refine the data structure alignment to produce a consensus sequence. The reasons for this design decision were twofold. First, in large sequencing projects, data is often produced over a period of weeks, or even months. As new data is produced the layout data structures can easily be updated. At any point, analysis can be done on the information at hand by using the analysis module, or building a consensus sequence that represents the existing data. Second, and perhaps more importantly, by separating the process of determining an initial alignment and the process of computing a consensus sequence, we allow several different methods for determining consensus sequences to be used and compared. This also permits the researcher to be more mathematically and computationally rigorous in stating the method used for producing consensus sequences.

Consensus finding is done using a multiway dynamic programming scheme. This is computationally feasible because the alignment done in constructing the layout data structures has reduced the dimensionality of the problem. The dynamic programming scheme that we use is similar to an *on-line algorithm*. We do not present the entire information from the data structure for alignment at one time. Instead a topological traversal of the data structure is used to divide fragments into subfragments based on the location of junction nodes in the data structure. The alignment is done in a left to right fashion with the algorithm forced to produce output soon after it processes each junction. Fragments can be moved left or right relative to their position in the data structure and gaps can be inserted, each subject to a penalty.

2.7 Statistical Analysis of the Sequence Data

The analysis package is a collection of statistical routines that were developed along side the assembly package ([11]). It handles input data in the form of layout data structures, as well as consensus sequence files. The tools provided are in two groups: tools for analyzing consensus sequences and tools for analyzing fragment assembly projects. For analyzing consensus sequences, modules are provided to compute: frame bias scores ([30]), Fickett scores ([7]), Staden scores ([28]), and Gribskov scores ([9]). In addition, dinucleotide frequencies can be analyzed within the sequence. For analyzing the sequencing project, modules are provided to compute the depth of fragment coverage and analyze the randomness of the fragment distribution. These modules interface with the GNU Plot graphing routines to produce graphical output.

3 Implementations and Experimental Results

All of our algorithms are implemented in the C programming language and run on platforms such as the SUN and DEC workstations. In addition, some of the modules have been implemented to run on a Sequent Symmetry multiprocessor. Using a DEC 3100 workstation, a project of 500 fragments can be assembled into a layout data structure in less than 15 minutes. The majority of this time is spent in the prepass and data structure construction modules.

In the subsections below, we describe one example assembly, as well as very briefly discuss our parallel implementations.

3.1 An Example Assembly

The appendix, which follows, contains a small section taken from an assembly. The first two pages show output produced by a topological traversal of the layout data structure. Since the fragments in this assembly spanned a region of several thousand bases, we

have shown only a short initial segment from the data structure. As the reader can easily see, the fragments align fairly well. However, two types of misalignments can be seen. First, when insertions or deletions have occurred in fragments these result in paths in the data structure of varying lengths. The topological traversal algorithm simply left justifies sequences (relative to the data structure's last junction point). Thus, misalignments are printed that are not present in the internal data structure representation. Second, since the alignment methods used to build the data structure are based on methods which consider only local regions of fragments, some true misalignments can occur. These are most commonly seen at the beginning, or the end, of a fragment. We rely on the multiway alignment package to correct these errors.

The second inclusion in the appendix is a small example of input and output for the multiway alignment algorithm, and an example of the message passing structure developed to implement dynamic programming as an on-line algorithm.

3.2 Parallel Implementations

To provide better interactive response, two phases of the assembly process have been implemented for the Sequent Symmetry multiprocessor: the prepass and the building of the data structures.

The parallelization of the prepass phase is a straightforward case of *data partitioning*, i.e., each processor executes the same (or very similar) code using different data. If we have n fragments and p processors, we assign about n/p fragments to each processor to build the bit vectors, and then each processor computes the entries of about n/p rows of the $n \times n$ matrix which holds the final answer.

Initial experimentation shows that the parallel version of the program exhibits good scaling properties, with the time for p processors being very close to the time for 1 processor divided by p.

The layout data structure construction algorithm was parallelized using a divide and conquer method that results naturally from the clustering phase of the assembly. Each cluster is assigned its own processor to construct the data structure. Unfortunately, this method somewhat restricts the amount of speedup that can be obtained – we can only expect the program to run as fast as it takes for one processor to build a data structure from the largest cluster of fragments. Furthermore, when the number of clusters exceeds the number of processors, it is necessary to implement a work queue.

Preliminary tests were run using two different groups of clusters constructed from *E. coli* sequencing projects. Each group of clusters was run on the Sequent in parallel, on the Sequent sequentially, and on a DECstation 3100 sequentially. The first group contained a variable number of fragments in each cluster, ranging from 2 fragments to 29 fragments. The second group of clusters had a more uniform number of fragments in each cluster, approximately 20 fragments for each cluster. The number of clusters ranged from 1 to 15.

Significant speedups were obtained for these data sets. However, the overhead for the algorithm is significant and we did not see the near linear speedup seen for the prepass algorithm. For the larger projects tested, the speedup was approximately a factor of 3.5.

4 Future Work

Several research projects remain. Further analysis of the maximum spanning tree algorithms is needed, we would like to present a theoretical model of the on-line version of dynamic programming, and finally parallel implementations on a CM-5 are currently under way.

Acknowledgements

We would like to thank Fred Blattner and the members of his lab in the Genetics Department at the University of Wisconsin.

References

[1] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. A basic local alignment search tool. *J. Mol. Biol.*, 215:403-410, (1990).

[2] D. Bacon and W. Anderson. Multiple sequence comparisons. *Methods Enzymol.*, 183:438-446, (1990).

[3] G. Churchil, C. Burks, M. Eggert, M. Engle, and M. Waterman. Assembling DNA sequence fragments by shuffling and simulated annealing. *Submitted for publication*, (1992).

[4] J. Devereux, P. Haeberli, and D. Smithies. A comprehensive set of sequence analysis programs for the VAX. *Nucl. Acids Res.*, 12:387-395, (1984).

[5] J. Dumas and J. Ninio. Efficient algorithms for folding and comparing nucleic acid sequences. *Nucl. Acids Res.*, 10:197-206, (1982).

[6] D. Feng and R. Doolittle. Progressive alignment and phylogenetic tree construction of aligned sequences. *Methods Enzymol.*, 183:375-387, (1990).

[7] J. Fickett. Recognition of protein coding regions in DNA sequences. *Nucl. Acids Res.*, 10:5303-5317, (1982).

[8] T. Gingerias, J. Milazzo, D. Sciaky and R. Roberts. Computer programs for assembly of DNA sequences. *Nucl. Acids Res.*, 7:529-545, (1979).

[9] M. Gribskov, J. Devereux and R. Burgess. *Nucl. Acids Res.*, 12:539-549, (1983).

[10] D. Joseph and J Meidanis. Hashing and back prediction methods for computing sequence similarity for fragment assembly. *To be submitted to SIAM Symposium on Discrete Algorithms,* (July 1992).

[11] A. Kryder. A Statistical package for genome sequencing. *Manuscript*, (1992).

[12] D. Lipman, S. Altschul, and J. Kececioglu. A tool for multiple sequence alignment. *PNAS USA*, 86:4412-4415, (1989).

[13] H. Martinez. An efficient method for finding repeats in molecular sequences. *Nucl. Acids Res.*, 11(13):4629-4634, (1983).

[14] W. Miller and E. Myers. Sequence comparison with concave weighting functions. *Bull. Math. Biol.*, 50(2):97-120, (1988).

[15] M. Murata. Three-way Needleman-Wunsch algorithm. *Methods Enzymol.*, 183:365-374, (1990).

[16] E. Myers and J. Kececioglu. *Unpublished.*

[17] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, 48:443-453, (1970).

[18] W. Pearson and D. Lipman. Improved tools for biological sequence comparison. *PNAS USA*, 85:2444-2448, (1988).

[19] W. Pearson. Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods Enzymol.*, 183:63-98, (1990).

[20] W. Pearson. Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics*, 11:635-650, (1991).

[21] H. Peltola, H. Söderland, and E. Ukkonen. SEQAID: A DNA sequence assembly program based on a mathematical model. *Nucl. Acids Res.*, 12:307-321, (1984).

[22] J. Posfai, A. Bhagwat, G. Posfai and R. Roberts. Predictive motifs derived from cytosine methyltransferases. *Nucl. Acids Res.*, 17:2421-2435, (1989).

[23] G. Schuler, S. Altschul and D. Lipman. A workbench for multiple alignment construction and analysis. *Proteins Struc. Func. Genet.*, 9:180-190, (1991).

[24] T. Smith and M. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195-197, (1981).

[25] E. Sobel and H. Martinez. A multiple sequence alignment program. *Nucl. Acids Res.*, 14(1):363-374, (1986).

[26] R. Staden. A strategy of DNA sequencing employing computer programs. *Nucl. Acids Res.*, 6:2601-2610, (1979).

[27] R. Staden. Automation of the computer handling of gel reading data produced by the shotgun method of DNA sequencing. *Nucl. Acids Res.*, 10:4731-4751, (1982).

[28] R. Staden and A. McLachlan. *Nucl. Acids Res.*, 10:141-156, (1982).

[29] W. Taylor. Hierarchical methods to align large numbers of biological sequences. *Methods Enzymol.*, 183:456-473, (1990).

[30] E. Uberbacher and R. Mural. Locating protein coding regions in human DNA sequences using a neural network – multiple sensor approach. *Manuscript,* (1991).

[31] M. Waterman and J. Griggs. Interval graphs and maps of DNA. *Bull. Math. Biol.*, 48(2):189-195, (1986).

[32] M. Waterman and R. Jones. Concensus methods for DNA and protein sequence alignment. *Methods Enzymol.*, 183:221-237, (1990).

[33] W. Wilbur and D. Lipman. Rapid similarity searches of nucleic acid and protein data banks. *PNAS USA*, 80:726-730, (1983).

A Layout Data Structure before Multiway Alignment

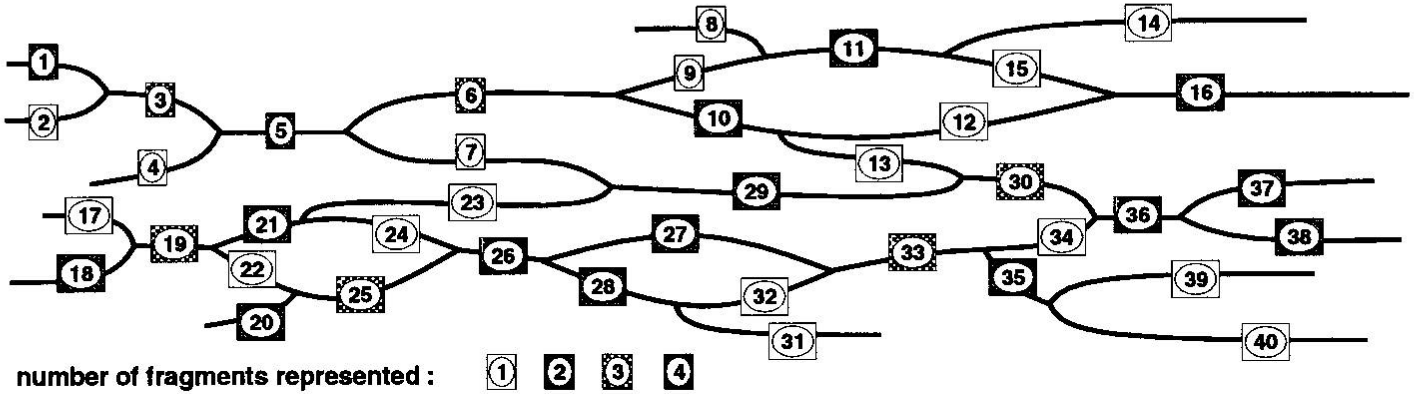
The layout data structre printed below uses two topological traversals. This topological sort positions junctions in the data structure relative to one another. Insertions and deletions induce shifts in the structure that may not align bases in the data structure. This is not a consensus alignment.

```
5-929:       ____________________________________________________________ ____________________________________________________________
68-946:      ____________________________________________________________ ____________________________________________________________
643-4489:    ____________________________________________________________ ____________________________________________________________
68b4344:     ____________________________________________________________ ____________________________________________________________
75-947:      ____________________________________________________________ ____________________________________________________________
5446-468-1:  ____________________________________________________________ ____________________________________________________________
599-4494:    ____________________________________________________________ ____________________________________________________________
494-969:     ____________________________________________________________ ____________________________________________________________
494-4482:    ____________________________________________________________ ____________________________________________________________
147-958:     ____________________________________________________________ ____________________________________________________________G
147-4524:    ____________________________________________________________ ____________________________________________________________
508b4479:    ____________________________________________________________ ____________________________________________________________GTC
53-946:      ____________________________________________________________ ____________________________________________________________TC
149-958:     TG__CTTCGTACACTT________CGCGGCGTGATTAGGA_TCGTGTCATCGACG_CTGC CG__TTCACGA_CGAGCCG_TTTCCGCAAAGTTACCGTAGACGACCGCCAGCCGGCCGTC
549-4480:    ____________________________________________________________ ____________________________________________________________
10-929:      TGGCTTTCGTACACTTTCACCGGGCGTGAAAATTTGAGGA_TACTGTCATCGACGCCTGC CGTTTTCACGATGCAGCCGTTTTCCGCAAAGTTACCGTAGAGCACCGCCAGCCGGCCGTC
2-929:       TGGCTTTCGTACACTTTCGCCGGGCGTGAAAATTTGAGGA_TACTGTCATCGACGCCTGC CGTTTTCACGACGCAGCCGTTTTCCGCAAAGTTACCGTAGA_SACCGCCAGCCGGCCGTC
435-964:     TGGCTTTCGTACACTTTCG_CGGGCGTGAAAATTTGAG__ATGCTGTCATCGA_GCCTGC CGTTTTCACGATGCAGCCGTTTTCCGCAAAGTTACCGTAGAGCACCGCCAGCCGGCCGTC
5435-468:    T_GCTTTCGTACACTTTCG_CGGGC__GTGAATTTGAG__ATGCTGTCATCGACGCCTGC CGTTTTCACGATGCAGCCGTTTTCCGCAAAGTTACCGTAGAGCACCGCCAGCCGGCXGTC
523b4479:    _____________________________TTTGAGGCATGCTGTCATCGACGCCTGC CGTTTTCACGATGCAGCCGTTTTCCGCAAAGTTACCGTAGAGCACCGCCA_GCCGCCGTC

5-929:       ____________________________________________________________ ____________________________________________________________
68-946:      ____________________________________________________________ ____________________________________________________________
643-4489:    ____________________________________________________________ ____________________________________________________________
68b4344:     ____________________________________________________________ ____________________________________________________________
75-947:      ____________________________________________________________ ____________________________________________________________
5446-468-1:  ____________________________________________________________ ____________________________________________________________
599-4494:    ____________________________________________________________ ____________________________________________________________
494-969:     ____________________________________________________________ ____________________________________________________________
494-4482:    ____________________________________________________________ ____________________________________________________________
147-958:     TTTGCTGTAGG_GTGTTCCAGCGAGCGGATACAGCCATTG_CGCGATCGTCGTTCAGCGT AT_CCAACGGC_AATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTGCAGGACCTGCGC
147-4524:    ____________________GAGC_GATACHGCCATTG_CGCGATCGT_GTCCAGCGT AT_CCAAC_GC_AATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTGCAGGACCTGCGC
508b4479:    TTTGCTGTAGGCGTGTTCCAGCGAGCGCATACAG_CATTGGCGCGATCGTCGTCCAGGGT ATCCCAACGGCTAATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTGCAGGACCTGCGC
53-946:      TTTGCTGTAG__CTGTTCCAGCGAGCG_ATACAG_CATTGGCGCGATCGTCGTCCAGCGT ATCCCAACGGC_AATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTCGAGGACCTGCGC
149-958:     TTTGCTGTAGGCGTGTTCCAGCGAGCGGATACAGCCATTGGCGCGATCGTCGTCCAGCGT ATCCCAACGGC_AATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTGCAGGACCTGCGC
549-4480:    ____________________________GCC_________ATCGTCGTCCAGCGT ATCCCAAC_GC_AATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTGCAGGACCTGCGC
10-929:      TTTGCTGTAGGCGTGTTCCAGCGAGCGGATACAG_CATTGGCGCGATCGTCGTCCAGCGT ATCCCAACGGC_AATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTGCAGGACCTGCGC
2-929:       TTTGCTGTAGGCGTGTTCCAGCGAGCGGATACAGCCATTGGCGCGATCGTCGTCCAGAGT ATCCCAACGGC_AATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTGCAGGACCTGCGC
435-964:     TTTGCTGTAGGCGTGTTCCA_CGAGCGGATACAGCCATTGGCGCGATCGTCGTCCAGCGT ATCCCAACGGC_AATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTGCAGGACCTGCGC
5435-468:    TTTGCTGTAGGXGTGTTCCAGCGAGCGGATACAGCCATTGGCGCGATCGTCGTCCAGCGT ATCCCAACGGC_AATCTGCGAGAATGGCGTGGTACGAATGCCTGCCTGCAGGACCTGCGC
523b4479:    TTTGCTGTAGGCGTGTTCCAGCGAGCGATGACAGCCATT_GCGCGGTCGTCGTCCAGCGT ATCCCAACAGC_AATC_TGCGAGAATGC____GTGGTACGAATGCCTGCAGGACCTGCGC

5-929:       ____________________________________________________________ ____________________________________________________________
68-946:      ____________________________________________________________ ____________________________________________________________
643-4489:    ____________________________________________________________ ____________________________________________________________
68b4344:     ____________________________________________________________ ____________________________________________________________
75-947:      ____________________________________________________________ _________________GTATGTTCCAGCGTTTGCGGCAACGATCAGGCCAAGACGTTTT
5446-468-1:  ____________________________________________________________ ____________________________________________________________
599-4494:    ____________________________________________________________ ____________________________________________________________TT
494-969:     ____________________________________________________________ _________________GATCCAGCGTTTGC_GCAA__GTAGGCCAAGTACTTTTCACA
494-4482:    _____________________________________ATAACGT_________________ ___________CGTATTGTTCCAGCGTTTGC_GCAACGTCAGGCCAAGTACTTTTCACA
147-958:     GGAACATATTTTTTACCGCGTCATCCTGGGTCAGCATAACGT_________________ ___________CGTATTGTTCCAGCGTTTGC_GCAACGTCAGGCCAAGTACTTTTCACA
147-4524:    GGAACATATTTTTTACCGCGTCATCCTGGGTCAGCATAACGT_________________ ___________CGTATTGTTCCAGCGTTTGC_GCAACGTCAGGCCAAGTACTTTTCACA
508b4479:    GGAACATATTTTTTACCGCGTCATCCTGGGTCAGCATAACGT_________________ ___________CGTATTGTTCCAGCGTTTGCGGCAACGTCAGGCCAAGTACTTTTCACA
53-946:      GGAACATATTTTTTACCGCGTCATCCTGGGTCAGCATAACGT_________________ ___________CGTATTGTTCCAGCGTTTGCGGCAACGTCAGGCCAAGCACTTTTCACA
149-958:     GGAACATATTTTTTACCGCGTCATCCTGGGTCAGCATAACGT_________________ ___________CGTATTGTTCCAGCGTTTGCGGCAACGTCAGGCCAAGTACTTTTCACA
549-4480:    GGAACATATTTTTTACCGCGTCATCCTGGGTCAGCATAACGT_________________ ___________CGTATTGTTCCAGCGTTTGC_GCAACGTCAGGCCAAGTAC__TTTTCA
10-929:      GGAACATATTTTTTACCGCGTCATCCTGGGTCAGCATAACGT_________________ ___________CGCATTGTTCCT_____________________________________
2-929:       GGAACATATTTTTTACCGCGTCATCC__________________________________ ____________________________________________________________
435-964:     GGAACATATTTTTTT_____________________________________________ ____________________________________________________________
5435-468:    GGAACATATTTTTTACCGCGTCA_____________________________________ ____________________________________________________________
523b4479:    GGAACATATTTTTTACCGCGTCATCCTGGGTCAGCATAACGTGTATGTCCAGCGTTTGCG GCAGCTCAGACAGTGCTTTTTCCT_____________________________________
```

```
5-929:       ____________________________________________________________ _________TGAACATCTTCCATATGTTATTTCTGGGTG_CTCGGCGCAACTTTACACAG
68-946:      ____________________________________________________________A CCAGCACGGTGAACATCTTCCATATGTTATTTCTGGGTG_CTCGGCGCAACTTTACACAG
643-4489:    ____________________CCCGCG_CGATCCAGTTCGCCGAGAATACCGATAACACCA CCAGCACGGTGAACATCTTCCATATGTTATTTCTGGGTG_CTCGGCGCAACTTTACACAG
68b4344:     ____________________________________________AACACCA CCAGCACGGTGAACATCTTCCATATGTTATTTCTGGGTG_CTCGGCGCAACTTTACACAG
75-947:      TCACATCATCGGTTCAGTAACCCGCG_CGATCCAGTTCGCCGTGAATACCGATAACACCA CCAGCACGGTGAACATC_TCCATATGTTATTTCTGGGTG_CTCGGCGCAACTTTACACAG
5446-468-1:  ____________________________________________________________ _____________________________ATTCTGGGT__STCGGCGCAACTTTACACAG
599-4494:    TCACATCACGGTTCAGTAACCCCGCGCCGATCCAGTTCGCCGAGAATACCGATAACACCA CCAGCACGGTGAACATCTTCCATATGTTATTTCTGGGTGCCTCGGCGCAACTTTACACAG
494-969:     TCACACG__GTTCAGTAA_CCCCGCG_CGATCCAGTTCGCCGAGAATACCGATAACACCA CCAGCACGGTGAACATCTTCCATATGTTATTTCTGGGTG_CTCGGCGCAACTTTACACAG
494-4482:    TCACACG__GTTCAGTAA_CCCCGCG_CGATCCAGTTCGCCGAGAATACCGATAACACCA CCAGCACGGTGAACATCTTCCATATGTTATTTCTGGGTG_CTCGGCGCAACTTTACACAG
147-958:     TCACACG__GTTCAGTAA_CCCCGCG_CGATCCAGTTCGCCGAGAATACCGATAACACCA CCAGCACGGTGAACATCTT_CATATGTTATTTCTGGGTG_CTCGGCGCAACTTTACACAC
147-4524:    TCACACG__GTTCAGTAA_CCCCGCG_CGATCCAGTTCGCCGAGAATACCGATAACACCA CCAGCACGGTGAACATC_T_CATAT_GTA_TTCTGGGTG_CTCGGCGCAACTTTACCA_G
508b4479:    TCACACGAGCTTCAGTAA_CCCCGCG_CGATCCAGTTCGCCGAGAATACCGATAACACCA CCAGCACGGTGAACATCTTCCATATGTTATTTCTGGGTG_CTCGGCGCAACTTT______
53-946:      TCACACG__GTTCAGTAA_CCCCGCG_CGATCCAGTTCGCCGAGAACACCGATAACACCA CCAGCACGGTGAACATCTTCCATATGTTATTTCTGGGT__STCGGCGCAACTTTACACAG
149-958:     TCACACG__GTTCAGTAA_CCCCGC_____________________________________ ____________________________________________________________
549-4480:    CATCACG__GTTCAGTAACCCCCGCG_CGATCCAGTTCGCCGAGAATACCGATAACACCA CCAGCACG_______TGACATTTCATTGATTTCTGG____CGTCGCGCA___CTTACACG
10-929:      ____________________________________________________________ ____________________________________________________________
2-929:       ____________________________________________________________ ____________________________________________________________
435-964:     ____________________________________________________________ ____________________________________________________________
5435-468:    ____________________________________________________________ ____________________________________________________________
523b4479:    ____________________________________________________________ ____________________________________________________________
```

B Multiway Alignment for a Layout Data Structure

layout data structure for a collection of fragments: obtained from the original sequence (shown below)



original sequence

CAGCG AAGGG AGAAA GAATT CACGG CCTAT GGCGC GAAGA ATTCC CCGAT TGCAG CACAT TATAA AACTT GGGTC CTTCC

input for multiway alignment program: obtained by two topological traversals of the data structure

```
1 CGGCGA                          8 AAGAAT              11 TGCCCGAAT    14 CCGCACATTATAAAATTTGGTC
   3 AGGGAGA       6 TCACGGCCTATGGCG       9 CCAAGATT                  15 CAGCACATTATA   16 AAACTTGGGTCCTT
2 CAGCCA       5 AAGAAT                          10 CTAGAAT      12 TGCCCAATTGCAGTATAA
   4 GGAACA       7 TTCACGCCCAATCG   29 CGCGAACATTTCCCCGATT   13 TCCCCGATTG          37 ACTTGGGTCCT
                     23 TTAACGGCCGATAG      27 TGTGGCGCGAAGAA     30 GCAGCACAT       38 AATCGGGTACTTCC
17 GAACG          21 GAAT       24 TCAGG      28 TATGGCGC                  34 GGAGA      36 GATAA
         19 AGAA    20 GAAAGA        26 GCCT       32 GAAGA    33 TTCCCGGATTGCA     39 AGATTATATAACTGGG
18 CAGCGAAGGG      22 CA      25 ATTCACG          31 GAAGAATTCACGGT      35 GC       40 ACATTCTAAAACTTGGGTCCTT
```

output from the multiway alignment program

```
    GAAGCG    AAGAAT      GCCT     CGCGAACATTTCCCCGATT    GGACGT       AATCGGGTACTTCC
CAGCGAAGGG      CA           TATGGCGC       TTCCCGGATTGCA       GATAA
CGGCGA     AGAA    ATTCACG            CCAAGATT          GCAGCACAT       ACTTGGGTCCT
CAGCCA      GAAAGA           TGTGGCGCGAAGAA TGCCCAATTGCAGTATAA     AAACTTGGGTCCTT
      AGGGAGA       TCAGG             GAAGA  TGCCCGAA-T     AGATTATATAACTGGG
        GGAACA     TTAACGGCCGATAG     AAGAAT           CAGCACATTATA
               GAAT                  CTAGAAT           CCGCACATTATAAAATTTGGTC
                  TTCACGCCCAATCG    GAAGAATTCACGGT       ACATTCTAAAACTTGGGTCCTT
                   TCACGGCCTATGGCG         TCCCCGATTG  GC
```

C Multiway Program Commands

```
SEQUENCE
id 1
weight 2
length 6
sequence CAGCGA

STARTPOINT
connect_seq_id 1
new_seq_id 1
location 0
left_fuzz 0
right_fuzz 0

SEQUENCE
id 2
weight 1
length 6
sequence CAGCCA

STARTPOINT
connect_seq_id 1
new_seq_id 2
location 0
left_fuzz 0
right_fuzz 0

SEQUENCE
id 18
weight 2
length 9
sequence CAGCGAAGG

STARTPOINT
connect_seq_id 1
new_seq_id 18
location 1
left_fuzz 1
right_fuzz 2

SEQUENCE
id 17
weight 1
length 5
sequence GAACG

STARTPOINT
connect_seq_id 1
new_seq_id 17
location 5
left_fuzz 2
right_fuzz 2

SEQUENCE
id 3
weight 3
length 7
sequence AGGGAGA

STARTPOINT
connect_seq_id 1
new_seq_id 3
location 6
left_fuzz 0
right_fuzz 0

SEQUENCE
id 4
weight 1
length 6
sequence GGAGAA

STARTPOINT
connect_seq_id 3
new_seq_id 4
location 1
left_fuzz 0
right_fuzz 2

SEQUENCE
id 19
weight 3
length 5
sequence GAGAA

STARTPOINT
connect_seq_id 18
new_seq_id 19
location 9
left_fuzz 0
right_fuzz 0

SEQUENCE
id 20
weight 2
length 9
sequence GGAGAAAGA

STARTPOINT
connect_seq_id 18
new_seq_id 20
location 7
left_fuzz 2
right_fuzz 2
```

```
SEQUENCE
id 5
weight 4
length 6
sequence AAGAAT

STARTPOINT
connect_seq_id 3
new_seq_id 5
location 7
left_fuzz 2
right_fuzz 2

SEQUENCE
id 21
weight 2
length 5
sequence GAATT

STARTPOINT
connect_seq_id 19
new_seq_id 21
location 6
left_fuzz 2
right_fuzz 2

SEQUENCE
id 22
weight 1
length 2
sequence CA

STARTPOINT
connect_seq_id 19
new_seq_id 22
location 5
left_fuzz 0
right_fuzz 0

SEQUENCE
id 25
weight 3
length 7
sequence ATTCACG

STARTPOINT
connect_seq_id 22
new_seq_id 25
location 2
left_fuzz 0
right_fuzz 0

SEQUENCE
id 6
weight 3
length 15
sequence TCACGGCCTATGGCG

STARTPOINT
connect_seq_id 5
new_seq_id 6
location 6
left_fuzz 2
right_fuzz 2

SEQUENCE
id 7
weight 1
length 14
sequence TTCACGCGCTATCG

STARTPOINT
connect_seq_id 5
new_seq_id 7
location 6
left_fuzz 2
right_fuzz 2

SEQUENCE
id 23
weight 1
length 13
sequence TAACGGCCTATGG

STARTPOINT
connect_seq_id 21
new_seq_id 23
location 5
left_fuzz 2
right_fuzz 2

SEQUENCE
id 24
weight 1
length 4
sequence CAGG

STARTPOINT
connect_seq_id 21
new_seq_id 24
location 5
left_fuzz 2
right_fuzz 2
```

```
SEQUENCE
id 26
weight 4
length 4
sequence GCCT

STARTPOINT
connect_seq_id 24
new_seq_id 26
location 4
left_fuzz 0
right_fuzz 0

SEQUENCE
id 27
weight 2
length 14
sequence TGTGGCGCGAAGAA

STARTPOINT
connect_seq_id 26
new_seq_id 27
location 4
left_fuzz 0
right_fuzz 0

SEQUENCE
id 28
weight 2
length 8
sequence TATGGCGC

STARTPOINT
connect_seq_id 26
new_seq_id 28
location 4
left_fuzz 0
right_fuzz 0

SEQUENCE
id 8
weight 1
length 11
sequence CCATGAAATTC

STARTPOINT
connect_seq_id 6
new_seq_id 8
location 16
left_fuzz 2
right_fuzz 2

SEQUENCE
id 31
weight 1
length 14
sequence GAAGAATTCACGGT

STARTPOINT
connect_seq_id 28
new_seq_id 31
location 8
left_fuzz 0
right_fuzz 0

SEQUENCE
id 32
weight 1
length 5
sequence GAAGA

STARTPOINT
connect_seq_id 28
new_seq_id 32
location 8
left_fuzz 0
right_fuzz 0

SEQUENCE
id 33
weight 3
length 13
sequence TTCCCGGATTGCA

STARTPOINT
connect_seq_id 27
new_seq_id 33
location 14
left_fuzz 0
right_fuzz 0

SEQUENCE
id 9
weight 1
length 10
sequence CCAAGATTTC

STARTPOINT
connect_seq_id 6
new_seq_id 9
location 15
left_fuzz 0
right_fuzz 0
```

```
SEQUENCE
id 10
weight 2
length 9
sequence CGATGAATT

STARTPOINT
connect_seq_id 6
new_seq_id 10
location 15
left_fuzz 0
right_fuzz 0

SEQUENCE
id 29
weight 2
length 19
sequence CGCGAACATTTCCCCGATT

STARTPOINT
connect_seq_id 7
new_seq_id 29
location 14
left_fuzz 0
right_fuzz 0

SEQUENCE
id 11
weight 2
length 7
sequence CCGATTG

STARTPOINT
connect_seq_id 8
new_seq_id 11
location 10
left_fuzz 2
right_fuzz 2

SEQUENCE
id 12
weight 1
length 18
sequence TCCCCGATTGCAGTATAA

STARTPOINT
connect_seq_id 10
new_seq_id 12
location 8
left_fuzz 2
right_fuzz 2

SEQUENCE
id 13
weight 1
length 9
sequence TGCCCGAAT

STARTPOINT
connect_seq_id 10
new_seq_id 13
location 8
left_fuzz 2
right_fuzz 2

SEQUENCE
id 14
weight 1
length 26
sequence CTGCACATTATAAAATTGGTCCTTCC

STARTPOINT
connect_seq_id 11
new_seq_id 14
location 7
left_fuzz 2
right_fuzz 2

SEQUENCE
id 15
weight 1
length 12
sequence CAGCACATTATA

STARTPOINT
connect_seq_id 11
new_seq_id 15
location 7
left_fuzz 0
right_fuzz 0

SEQUENCE
id 30
weight 3
length 8
sequence GCAGCACA

STARTPOINT
connect_seq_id 13
new_seq_id 30
location 8
left_fuzz 2
right_fuzz 2
```

```
SEQUENCE
id 34
weight 1
length 5
sequence GGAGA

STARTPOINT
connect_seq_id 33
new_seq_id 34
location 13
left_fuzz 0
right_fuzz 0

SEQUENCE
id 35
weight 2
length 6
sequence GCACAT

STARTPOINT
connect_seq_id 33
new_seq_id 35
location 13
left_fuzz 0
right_fuzz 0

SEQUENCE
id 36
weight 4
length 3
sequence TCA

STARTPOINT
connect_seq_id 30
new_seq_id 36
location 8
left_fuzz 2
right_fuzz 2

SEQUENCE
id 16
weight 2
length 10
sequence AACTTGGGTC

STARTPOINT
connect_seq_id 15
new_seq_id 16
location 12
left_fuzz 0
right_fuzz 0

SEQUENCE
id 39
weight 1
length 14
sequence TATATAAAACTGGG

STARTPOINT
connect_seq_id 35
new_seq_id 39
location 6
left_fuzz 0
right_fuzz 0

SEQUENCE
id 40
weight 1
length 19
sequence TCTAAAACTTGGGGTCCTT

STARTPOINT
connect_seq_id 35
new_seq_id 40
location 6
left_fuzz 0
right_fuzz 0

SEQUENCE
id 37
weight 2
length 14
sequence TAAAATTGGGTCCT

STARTPOINT
connect_seq_id 36
new_seq_id 37
location 3
left_fuzz 0
right_fuzz 0

SEQUENCE
id 38
weight 2
length 17
sequence TTAAATCGGGTACTTCC

STARTPOINT
connect_seq_id 36
new_seq_id 38
location 3
left_fuzz 2
right_fuzz 2
```